

1       SYSTEM AND METHOD FOR ASSISTING IN THE DEVELOPMENT AND  
2               INTEGRATION OF REUSABLE CIRCUIT DESIGNS

3               Carol A. Fields

4               Anthony D. Williams

5  
6       FIELD OF THE INVENTION

7               The present invention generally relates to electronic  
8       design tools, and more particularly to assisting in the  
9       creation and usage of design modules that are amenable for  
10      reuse in other designs.

11  
12      BACKGROUND

13              Increasingly complex functions are being implemented in  
14      ASIC and field programmable gate arrays (FPGAs) given recent  
15      advances in silicon technology. Design trends are shifting  
16      toward system-level integration in order to reduce the time-  
17      to-market and reduce development costs.

18              System-level integration relies on reuse of previously  
19      created designs, either from within an enterprise or from a  
20      commercial provider. The engineering community sometimes  
21      refers to these previously created designs as " design  
22      modules" , " cores" or " IP" (intellectual property). As  
23      on-chip processors and large functional blocks become  
24      increasingly common, vendors are making complex design  
25      modules available for general usage, and companies are  
26      making modules for reuse within the respective  
27      organizations. These design modules are then integrated  
28      into larger systems by end-users.

29              For reusable design modules to be successful, they must  
30      be easy to use. Design modules that are amenable to reuse  
31      must be well documented, possess sufficient  
32      parameterization, have an understandable structure, follow  
33      certain coding guidelines, and be extensible for future  
34      usage. Reusable design modules should also have a well-  
35      planned and documented testbench. To varying degrees, many  
36      commercial and internal design modules satisfy these

1 objectives. However, the process and tools used to create a  
2 design module will often limit the ease and extent to which  
3 the objectives can be satisfied.

4 A method that address the aforementioned problems, as  
5 well as other related problems, is therefore desirable.

6  
7 SUMMARY OF THE INVENTION

8 In various embodiments, the invention provides a method  
9 and system for developing a reusable electronic circuit  
10 design module and using the design module in a debug mode.

11 In one embodiment, the functional design elements comprising  
12 a design module are entered into a database along with  
13 documentation elements that describe the design elements.

14 The functional design elements are linked with selected ones  
15 of the documentation elements in the database. A testbench  
16 is simulated with the design module, and the generated  
17 results are stored in a database and linked with the  
18 functional design elements. By linking the design elements,  
19 documentation, translation results, and simulation results,  
20 the characteristics of the design module are easily  
21 ascertained by a designer who is reusing the design module.

22 In another embodiment, a system includes a database, a  
23 design inspector, a debugging-support module, and a  
24 functional simulator. The database is arranged for storage  
25 of the design elements and documentation elements, and the  
26 design inspector is coupled to the database. The design  
27 inspector links the functional design elements with  
28 selected ones of the documentation elements. The  
29 debugging-support module is coupled to the simulator and to  
30 the database, and generates a netlist from the design  
31 module, wherein the netlist is suitable for simulation.  
32 The functional simulator is coupled to the debugging-  
33 support module and simulates a testbench with the design  
34 module, whereby simulation results are generated. The

1 simulation results are entered in the database by the  
2 debugging-support module and thereafter linked with the  
3 design elements.

4 The above summary of the present invention is not  
5 intended to describe each disclosed embodiment of the  
6 present invention. The figures and detailed description  
7 that follow provide additional example embodiments and  
8 aspects of the present invention.

9  
10 BRIEF DESCRIPTION OF THE DRAWINGS

11 Various aspects and advantages of the invention will  
12 become apparent upon review of the following detailed  
13 description and upon reference to the drawings in which:

14 FIG. 1 is a block diagram of a system for creating  
15 reusable design modules in accordance with one embodiment of  
16 the invention;

17 FIG. 2 is a flowchart of an process for developing  
18 reusable logic in accordance with one embodiment of the  
19 invention; and

20 FIG. 3 is a flowchart of a process for integrating  
21 reusable logic into a design in accordance with another  
22 embodiment of the invention.

23  
24 DETAILED DESCRIPTION

25 The present invention is believed to be applicable to  
26 the process of creating reusable design modules for a  
27 variety of electronic circuit technologies. An example  
28 environment for creating design modules for field  
29 programmable gate arrays (FPGAs) is used to describe the  
30 various embodiments of the invention. While the present  
31 invention is not so limited, an appreciation of the present  
32 invention is presented by way of specific examples involving  
33 FPGAs.

34 In accordance with various embodiments of the  
35 invention, a design inspector tool works in conjunction with  
36 a design entry tool to assist in creating design modules

1 that are easy to reuse. Specifically, the design inspector  
2 processes a design module to determine whether there is  
3 documentation that conforms to specified criteria and  
4 whether the design module conforms to other specified design  
5 rules. Where the design module fails to conform to the  
6 specified criteria, a report is provided so that the  
7 deficiency can be remedied. In addition, the design  
8 elements that comprise the design module, the associated  
9 documentation, translation results, and simulation results  
10 are linked in a database to assist in understanding  
11 particular aspects of the design in view of simulation  
12 results.

13 The present invention supports two modes of operation.  
14 The first mode is a design mode where the first instance of  
15 a design module is created for reuse. The initial design  
16 module may or may not be part of a fully operational system  
17 or integrated with other modules. The second mode is an  
18 integration mode where a reusable design module is  
19 integrated with other design modules to create a netlist.  
20 The netlist is translated and then simulated, wherein the  
21 translation results and the simulation results are  
22 correlated with the design modules and associated  
23 documentation. After creating a physical implementation  
24 from the netlist, the physical implementation is simulated,  
25 and the simulation results are correlated with the design  
26 modules and associated documentation. The final design, as  
27 well as the design module as modified, can be saved in a  
28 central depository for further reuse.

29 By inspecting for desired documentation and desired  
30 design characteristics at appropriate stages and creating a  
31 database of design modules, documentation, netlist, and  
32 simulation results, easy-to-reuse design modules can be  
33 created.

34 FIG. 1 is a block diagram of a system for creating  
35 reusable design modules in accordance with one embodiment of  
36 the invention. System 100 includes database 102, which

1 includes various design elements and associated  
2 documentation elements. Design inspector 104, when  
3 activated by the user, examines the design elements for  
4 various predefined characteristics. For example, inspector  
5 104 inspects for proper documentation, along with desirable  
6 parameterization, security, revision control, hierarchical  
7 arrangement, partitioning, and adherence to predetermined  
8 design rules, all as embodied in logic within the design  
9 inspector.

10 Design entry tool 106 is used to initially create a  
11 design module, and in different embodiments may be an RTL  
12 editor, state machine editor, or schematic capture tool, for  
13 example. The user interacts with the various components via  
14 user interface logic 108, and design inspector 104 can be  
15 invoked either via design entry tool 106 or by the user via  
16 user interface 108.

17 Database 102 is created by design entry tool 106 via  
18 design inspector 104. Database 102 includes several design  
19 elements 110 that comprise a design module. Associated with  
20 the design elements are various documentation elements 112.  
21 The documentation elements may include, among other items, a  
22 datasheet, a functional block diagram, a state diagram,  
23 descriptions of blocks, intended usage, parameters that can  
24 be modified, effects of modifying parameters, expansion  
25 effects, descriptions of input and output signals,  
26 description of processing as well as other design  
27 descriptive information.

28 Database 102 not only provides a mechanism to reference  
29 documentation associated with various elements of a design,  
30 but also provides a mechanism to correlate the design  
31 elements and associated documentation with a netlist 114 and  
32 physical implementation 116, along with the functional  
33 simulation results 118 and physical simulation results 120.  
34 Since a design module will undergo various translations,  
35 e.g., synthesis, in progressing toward simulation, the  
36 translation results are correlated with design elements and

1 associated documentation in order to facilitate debugging a  
2 design. Thus, based on the simulation results, a designer  
3 can easily reference design information and documentation  
4 associated with the implementations.

5 When a designer is ready to begin testing and debugging  
6 a design, a netlist 114 is created. The netlist, along with  
7 a suitable testbench (not shown) is used to test the  
8 functionality of the design. The testbench is stored in a  
9 file which is correlated with the design elements. The  
10 design is simulated for functional correctness using  
11 functional simulator 122 and applying the testbench. Error  
12 and warning messages are written to the database and  
13 correlated with the design by debug support logic 124.  
14 Example functional simulators include Modelsim from Model  
15 Technology and VSS from Synopsys. The data resulting from  
16 the simulation is written to functional simulation results  
17 file 118.

18 In order to facilitate debugging, debugging support  
19 logic 124 correlates the simulation results from functional  
20 simulator 122 with design elements 110 and documentation  
21 elements 112 from database 102. For example, if design  
22 entry tool 106 is a state machine editor, errors are  
23 correlated to possible state or state transitions containing  
24 the error. For a schematic capture design entry tool, a  
25 correlation of the error to the vicinity of the schematic  
26 containing the error is provided. Correlation of simulation  
27 results to design elements and documentation elements  
28 enables display of the documentation via user interface 108.

29 Debugging support logic 124 tracks and correlates how  
30 the design module is translated. For example, HDL design  
31 constructs are traced from high-level design to design  
32 elements. In schematic C, any changes made by a translation  
33 tool are tracked.

34 After the errors discovered in the functional  
35 simulation have been corrected, the design can be physically  
36 implemented via implementation translators 128. Example

1 translators include DC2NCF, NGD2VHDL, and NGDZVER  
2 translators from Xilinx. Physical implementation 116 is a  
3 netlist, for example.  
4 Physical simulator 126 runs a simulation using a predefined  
5 testbench and interfaces with debugging support logic 124 to  
6 log the physical simulation results 120. The physical  
7 simulation results are also correlated with the design  
8 elements and documentation of database 102. By tracking the  
9 translation of the design elements from high level design  
10 through translation to the physical implementation, the  
11 constructs of the high-level design are correlated with  
12 elements in the physical implementation. This correlation  
13 is then used by debugging support logic 124 to correlate the  
14 physical simulation results to the design and documentation  
15 elements. In FPGA technology, the representation of the  
16 design may differ from the implementation. Tracking the  
17 changes as the design elements progress through the  
18 translations and correlating the changes to the  
19 documentation assists in reuse of a module.

20 FIG. 2 is a flowchart of a process for developing  
21 reusable logic in accordance with one embodiment of the  
22 invention. The process generally entails entering and then  
23 inspecting a design module, modifying the design to correct  
24 any deficiencies and then re-inspecting. The design module  
25 is also inspected for a proper level of documentation. The  
26 design elements that comprise the design module and  
27 documentation elements that are associated with the design  
28 elements are stored in a database for future reference. In  
29 addition to the design module, a testbench is created for  
30 use with the simulator. The testbench is also associated  
31 with the design module for future use.

32 At step 202, a design module is created using a design  
33 entry tool that is adapted to provide the functions of the  
34 present invention. A design script is also created by the  
35 designer. The design script contains the directives that  
36 specify which tools to run, the order in which the tools are

1 to run, as well as options and environment variables for the  
2 tools. The design script is stored in a separate file.

3 The design module and script file are inspected at step  
4 204 for selected design characteristics. For example, the  
5 characteristics may include desired parameterization,  
6 adherence to specified design rules, and a suitable  
7 hierarchical arrangement of the design. In addition, the  
8 design inspector checks that all ranges are enumerated and  
9 that there is a consistent number of multi-bit objects.

10 The design inspector is configured to enforce certain  
11 design rules. For example, the rules may include a certain  
12 number of spaces for indentation of the code, one statement  
13 per line, a maximum of 72 characters/line, use of tabs for  
14 tabular layout, no usage of abbreviations, capitalization  
15 rules, usage of suffixes, reserved uppercase for constants,  
16 usage of underscore character for separation of compound  
17 words, no misuse of reserved words, usage of "\_n" for  
18 active low symbols, usage of "clk" prefix for clock  
19 signals, usage of a common clock name for all derived and  
20 synchronized clock signals, usage of symbolic names to  
21 define states in a finite state machine, proper usage of  
22 filename extensions for identification of file type, and  
23 language specific design rules (e.g., process labels).

24 For a desired hierarchical arrangement, the design  
25 inspector checks whether there are constants that can be  
26 changed to variables defined at the top sub-module level,  
27 and that there are no more than three levels of nesting. In  
28 addition, the design inspector checks that names are  
29 preserved across interfaces and hierarchical boundaries.  
30 The design inspector may also be programmed to provide a  
31 graphical representation of the present hierarchical  
32 arrangement to assist in tracking, adding, and deleting sub-  
33 modules.

34 Where the design module fails to conform to the  
35 selected characteristics, a report is provided to the  
36 designer at step 206. In response to the report by the

1 design inspector, the user may modify the design and then  
2 re-inspect (step 208). The modify and re-inspect cycle may  
3 be repeated as many times as required to achieve desired  
4 design goals.

5 As part of the re-inspection, the design inspector  
6 tracks which of the design elements are changed from the  
7 prior inspection and reports which additional design  
8 elements are dependent on such changes. For example,  
9 changing the value of a constant causes the design inspector  
10 to report all sub-modules which use the constant.

11 At step 210, the design inspector is invoked to check  
12 that documentation has been entered for the design module  
13 and that the documentation conforms to characteristics  
14 imposed by the design inspector. At step 212, the  
15 documentation can be entered and/or updated in response to  
16 the report provided by the design inspector. Thereafter the  
17 design modules can be re-inspected.

18 There are numerous types of documentation that may be  
19 required. Examples include: copyright and confidentiality  
20 notices, brief descriptions, revision numbers, past and  
21 current authors, change histories, functionality  
22 descriptions, descriptions of code structure, variable  
23 definitions and side effects, enumeration of valid input  
24 data ranges, identifications of parameters not intended to  
25 be easily modified, and cross references to applicable  
26 industry standards. In addition it may be desirable to  
27 require documentation as to the implementation implications  
28 of modifying various parameters. For example, expanding a  
29 16x16 multiplier may cause the multiplier to wrap into  
30 several columns of configurable logic blocks (CLBs) in a  
31 field programmable gate array (FPGA).

32 Once the desired documentation has been entered, the  
33 documentation elements and design elements that comprise  
34 the design module are linked in a database. The database  
35 provides easy future reference to the design and

1 documentation, such as when the design has been implemented  
2 and has undergone several translation and simulation steps.

3 At step 216, a testbench is entered by the user using  
4 conventional tools. The testbench will also be referred to  
5 in this document as "simulation elements". Simulation  
6 elements are similar to design elements except that they are  
7 generated for the purpose of testing the functionality of  
8 the design. The testbench is inspected at step 218 by the  
9 design inspector, and at step 220, the characteristics of  
10 the testbench are reported to the user. The testbench is  
11 modified, as may be necessary, at step 222 and then re-  
12 inspected. The modify/re-inspect step may be repeated as  
13 necessary to correct any deficiencies.

14 At step 224, the documentation for the testbench is  
15 created, and the testbench is re-inspected. Example  
16 documentation for testbenches includes: documenting the  
17 features included in the testbench, enumeration of  
18 assumptions and omissions, description of an input sequence,  
19 description of expected output behavior, and a description  
20 of anticipated design flow. The elements that comprise the  
21 testbench and the associated documentation are added to the  
22 design database at step 226. Once the design and testbench  
23 have been suitably structured and documented and added to  
24 the database, the process is complete. The design module is  
25 then in a form that is amenable to reuse.

26 FIG. 3 is a flowchart of a process for integrating  
27 reusable logic into a design in accordance with another  
28 embodiment of the invention. The process generally entails  
29 integrating a design module, which was created in accordance  
30 with the process of FIG. 2, with other design modules to  
31 create a netlist. The resulting logic can then be  
32 documented, structured, and inspected to create a design  
33 module that can be saved for future integration with still  
34 other design modules.

35 At step 302, the desired design module is retrieved  
36 from a central depository. The central depository may be a

1 tree structure of files containing design modules,  
2 associated documentation, and test benches. The  
3 documentation elements and testbench associated with the  
4 selected design module are retrieved at step 304.

5 The selected design module is modified at step 306 in a  
6 manner that is suitable for integration with other design  
7 modules. The nature of the modifications to the selected  
8 design module is dependent on the function and interfaces of  
9 the modules with which the selected module is being  
10 integrated. The documentation elements and testbench are  
11 also modified as may be necessary.

12 At step 308, the new design (selected design module as  
13 integrated with other design modules) is translated into a  
14 netlist. The netlist is then written to a file at step 310.  
15 At step 312, the translation results are correlated with the  
16 design elements, documentation elements, and testbench  
17 elements of the central depository database. In order to  
18 correlate elements of the netlist with the original design  
19 elements, the elements generated during the translation  
20 process are associated in a database with the respective  
21 design elements from which they were generated.

22 The functional design is simulated at step 314, and the  
23 simulation results are written to a file at step 316. At  
24 step 318, the simulation results are correlated with the  
25 design elements, documentation elements, and testbench  
26 elements in the central depository database. The  
27 correlation provides a mechanism for the designer to trace  
28 particular portions of the simulation results back to the  
29 original design elements and associated documentation. If  
30 an error is discovered during functional simulation, the  
31 offending design elements can be changed, saved, re-  
32 implemented, and simulated as needed.

33 At step 320, the functional design is translated into a  
34 physical design using conventional tools adapted to work in  
35 conjunction with the present invention. The physical  
36 implementation is written to a file at step 322, and the

1 physical translation results are correlated with the design  
2 elements and documentation elements at step 324.

3 The following example illustrates the correlation of  
4 design elements to physical translation results. The  
5 following snippet of VHDL code is taken from a design that  
6 implements a state machine.

```
7     constant IDLE_CNT    : INTEGER := 8;  
8     ...  
9  
10    when STABILIZATION_WAIT =>  
11    ISOLATE               <= TRUE;  
12    TIMER_TICK           <= CLK_1K;  
13    if(IDLE_DONE = TRUE) then  
14        NEXT_STATE <= LINK_WAIT;  
15    elseif (SCARRIER_PRESENT = TRUE) then  
16        NEXT_STATE <= VALID_START;  
17    else  
18        NEXT_STATE <= STABILIZATION_WAIT;  
19    endif;  
20    ...  
21    process (COUNT)  
22    begin  
23        if (COUNT < IDLE_CNT) then  
24            IDLE_DONE <= '0';  
25        else  
26            IDLE_DONE <= '1';  
27        end if;  
28    end process;  
29    ...
```

30 In documenting this portion of the design, the designer  
31 creates documentation for the design element  
32 STABILIZATION\_WAIT indicating that a constant IDLE\_CNT  
33 controls the variable IDLE\_DONE, which influences what the  
34 transition to the LINK\_WAIT state. Further documentation  
35 that is associated with IDLE\_DONE explains the usage of  
36 IDLE\_DONE and the implications of changing the constant  
37 value.

38 The following snippet of code sets forth the physical  
39 translation results that is generated from the VHDL set  
40 forth above.

```
41    defparam \current_state(0)/G .INIT = 16'hECCC;  
42    X_LUT4 \current_state(0)/G (  
43        .ADR0 (un26_count_3),  
44        .ADR1 (current_state[6]),  
45        .ADR2 (current_state[0]),
```

```

1      .ADR3 (SCARRIER_PRESENT_c),
2      .0 (\current_state[0]/GROM )
3  );
4  defparam \current_state(0)/F .INIT = 16'h0001;
5  X_LUT4 \current_state(0)/F (
6      .ADR0 (current_state[1]),
7      .ADR1 (current_state[6]),
8      .ADR2 (current_state[2]),
9      .ADR3 (current_state[0]),
10     .0 (\current_state[0]/FROM )
11 );
12 X_BUF \current_state(0) /XUSED (
13     .I (\current_state[0]/FROM ),
14     .O (ISOLATE_iv)
15 );
16 X_FF \current_state(0)/FFY/ASYNC_FF (
17     .I (\current_state[0]/GROM ),
18     .CLK (RIC_CLK_c),
19     .CE (VCC),
20     .SET (GND),
21     .RST (\current_state[0]/FFY/ASYNC_FF_GSR_OR ),
22     .O (current_state[0])
23 );
24

```

It can be seen in the physical translation that the state STABILIZATION\_WAIT has been renamed. Without correlation between the physical translation results and the functional design element, the designer would be left to determine which elements in the physical translation correspond to the elements in the design. In this example, current\_state[0] corresponds to STABILIZATION\_WAIT. To assist the designer during test and debug activities, STABILIZATION\_WAIT is correlated with current\_state[0] by the translators and debugging support logic. Thus, when performing physical simulation, results that reference current\_state[0] can be traced by the designer to the state STABILIZATION\_WAIT, which has linked documentation that references the constant IDLE\_CNT.

The correlation of the physical translation results with the original design elements and documentation elements is especially helpful in the context of designs for programmable logic devices (PLDs). Example PLDs include field programmable gate arrays (FPGAs) that are available from Xilinx. The FPGA-based physical implementation of a

1 design may have little or no resemblance to the original  
2 design module. Thus, it is beneficial to correlate elements  
3 of the physical translation with the originating design  
4 elements and associated documentation elements.

5 The physical implementation is simulated at step 326,  
6 and the simulation results are written to a file at step  
7 328. At step 330, the simulation results are correlated  
8 with the design elements and documentation elements. If  
9 redesign is necessary, the appropriate design elements can  
10 be modified, and the process can be repeated beginning at  
11 step 308.

12 At step, 332, the new design is subjected to the  
13 process of FIG. 2. That is, the new design is processed by  
14 the design inspector to determine whether the selected  
15 design rules and documentation requirements have been  
16 adhered to.

17 Accordingly, the present invention provides, among  
18 other aspects, a system and method for creating reusable  
19 design modules for electronic circuits. Other aspects and  
20 embodiments of the present invention will be apparent to  
21 those skilled in the art from consideration of the  
22 specification and practice of the invention disclosed  
23 herein. It is intended that the specification and  
24 illustrated embodiments be considered as examples only, with  
25 a true scope and spirit of the invention being indicated by  
26 the following claims.

27